Attorney Docket No. SUN1P829/P6115

PATENT APPLICATION

LIGHTWEIGHT NATIVE METHOD INVOCATION INTERFACE FOR JAVA COMPUTING ENVIRONMENTS

Inventors: 1. Stepan Sokolov
34832 Dorado Common
Fremont, CA 94555
Citizenship: Ukraine

2. David Wallman
777 S. Mathilda Ave., #266
Sunnyvale, CA 94087
Citizenship: Israel

Assignee: Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303

BEYER WEAVER & THOMAS, LLP
P.O. Box 778
Berkeley, CA 94704-0778
Telephone (650) 961-8300

LIGHTWEIGHT NATIVE METHOD INVOCATION INTERFACE FOR JAVA COMPUTING ENVIRONMENTS

BACKGROUND OF THE INVENTION

The present invention relates generally to object-based high level programming environments, and more particularly, to techniques for invoking native methods in Java computing environments.

Recently, the Java programming environment has become quite popular. The Java programming language is a language that is designed to be portable enough to be executed on a wide range of computers ranging from small devices (e.g., pagers, cell phones and smart cards) up to supercomputers. Computer programs written in the Java programming language (and other languages) may be compiled into Java Bytecode instructions that are suitable for execution by a Java virtual machine implementation.

The Java virtual machine is commonly implemented in software by means of an interpreter for the Java virtual machine instruction set but, in general, may be software, hardware, or both. A particular Java virtual machine implementation and corresponding support libraries together constitute a Java runtime environment.

Computer programs in the Java programming language are arranged in one or more classes or interfaces (referred to herein jointly as classes or class files). Such programs are generally platform, i.e., hardware and operating system, independent. As such, these computer programs may be executed without modification on any computer that is able to run an implementation of the Java runtime environment.

Object-oriented classes written in the Java programming language are compiled to a particular binary format called the "class file format." The class file includes various components associated with a single class. These components can be, for example, methods and/or interfaces associated with the class. In addition, the class file format can include a significant amount of

ancillary information that is associated with the class. The class file format (as well as the general operation of the Java virtual machine) is described in some detail in The Java Virtual Machine Specification, Second Edition, by Tim Lindholm and Frank Yellin, which is hereby incorporated herein by reference.

A virtual machine operating in a Java computing environment allows for execution of native methods (functions or procedures) written in platform specific programming languages (native languages). In other words, the Java programming language allows invocation of native methods written in other programming languages (e.g., C, C++, etc.). This means that the processing of the native method is performed by a section of code written in the native programming language. Accordingly, there is a need to convert the parameters (e.g., Java primitive data types, Java reference objects) from Java programming language to parameters suitable for the native language.

To achieve this, recently, Java Native Interface (JNI) has been developed as an interface between the Java program and the native methods written in programming languages other than Java (e.g., C, C++, etc.). Using Java Native Interface (JNI), native methods are provided a reference (environment parameter). More particularly, every invocation of a native method passes a reference to a table of functions that are used to facilitate the conversion of parameters from Java to the native language. Thus, the Java Native Interface (JNI) does not give direct access to Java parameters (e.g., Java primitive data types, Java reference objects). Instead, these parameters are accessed and indirectly through various Java Native Interface (JNI) methods which perform the conversation. More details about the Java Native Interface (JNI) are provided in Core Java 2, Volume II-Advanced Features, by Cay S. Horstmann and Gary Cornell, which is incorporated by reference herein for all purposes.

Although the Java Native Interface (JNI) provides a useful tool for invocation of native programs in Java computing environments, there is a significant cost associated with using the Java Native Interface (JNI). In other words, there is a significant amount of memory and runtime overhead associated with the use of the methods of the Java Native Interface (JNI). As a

result, using the Java Native Interface (JNI) can have adverse effects on the performance of virtual machines, especially those operating with relatively less memory and/or computing power (e.g., embedded systems).

In view of the foregoing, there is a need for alternative techniques for invocation of native methods in Java computing environments.

SUMMARY OF THE INVENTION

Broadly speaking, the present invention relates to improved techniques for invocation of native methods in Java computing environments. These techniques can be implemented in Java computing environments to facilitate use of methods (functions or subroutines) written in programming languages other than Java (e.g., C, C++, etc.). As will be appreciated, these techniques are highly suitable for use by virtual machines operating with relatively less memory and/or computing power (e.g., embedded systems).

In accordance with one aspect of the invention, a lightweight native method invocation interface is disclosed. In one embodiment, the lightweight native method invocation provides direct access to Java parameters on the execution stack. In addition, the lightweight native method invocation can include macro instructions that operate efficiently to convert the Java parameters into native parameters. Thus, the lightweight native method invocation can significantly reduce the overhead associated with conventional Java native method invocation techniques. As a result, performance of virtual machines, especially those operating with relatively less memory and/or computing power, can be improved.

The invention can be implemented in numerous ways, including as a method, an apparatus, a computer readable medium, and a database system. Several embodiments of the invention are discussed below.

As a method for invoking a native method written in a programming language other than Java, one embodiment of the invention includes the acts of: providing a reference to one or more Java parameters on a Java execution stack, the one or more Java parameters being parameters associated with the native method; and accessing at least one of the one or more Java parameters based on the reference.

As a method for invoking a native method written in a native programming language other than Java, one embodiment of the invention includes the acts of: providing a reference to one or more Java parameters associated with the native method on a Java execution stack; accessing at least

one of the one or more Java parameters based on the reference; converting at least one Java parameter to a native parameter suitable for the native programming language. The converting is performed by a set of macro instructions written in the native programming language.

As a Java program, one embodiment of the invention includes a native method invocation instruction operating to invoke a native method written in a programming language other than Java. The method invocation instruction operates to provide a reference to an entry on an execution stack. The entry is a reference to at least one parameter associated with the method invocation instruction; and the reference is provided directly to a set of macro instructions.

As a computer readable media, one embodiment of the invention includes computer program code for a lightweight native Java method invocation interface operating in a Java computing environment to facilitate invocation of native methods written in a programming language other than Java. The computer program code includes code for a set of macro instructions. In addition, a reference to an execution stack is provided to a set of macro instructions. The reference provides a reference to at least one parameter associated with the method invocation instruction which is on the execution stack.

These and other aspects and advantages of the present invention will become more apparent when the detailed description below is read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

Fig. 1 represents a Java programming environment in accordance with one embodiment of the invention.

Fig. 2 represents a Java computing environment including a Java execution stack and a native execution stack.

Fig. 3 illustrates a method for invocation of Java native methods in accordance with one embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention pertains to improved techniques for invocations of native methods in Java computing environments. These techniques can be implemented in Java computing environments to facilitate use of methods (functions or subroutines) written in programming languages other than Java (e.g., C, C++, etc.). As will be appreciated, these techniques are highly suitable for use by virtual machines operating with relatively less memory and/or computing power (e.g., embedded systems).

In accordance with one aspect of the invention, a lightweight native method invocation interface is disclosed. In one embodiment, the lightweight native method invocation provides direct access to Java parameters on the execution stack. In addition, the lightweight native method invocation can include macro instructions that operate efficiently to convert the Java parameters into native parameters. Thus, the lightweight native method invocation can significantly reduce the overhead associated with conventional Java native method invocation techniques. As a result, performance of virtual machines, especially those operating with relatively less memory and/or computing power, can be improved.

Embodiments of the invention are discussed below with reference to Figs. 1-3. However, those skilled in the art will readily appreciate that the detailed description given herein with respect to these figures is for explanatory purposes only as the invention extends beyond these limited embodiments.

Fig. 1 represents a Java programming environment 100 in accordance with one embodiment of the invention. The Java programming environment 100 can be implemented by a virtual machine operating in a Java computing environment. As shown in Fig. 1, a Java application program (or applet) 102 uses a lightweight native method invocation interface 104 to access one or more native methods 106 written in a non-Java programming language (e.g., C, C++, etc.).

In the described embodiment, the lightweight native method invocation interface 104 includes a native macro portion 108 which provides the native methods 106 with access to the Java application program (or applet) 102. The native macro portion 108 represents computer instructions typically written in the same programming language as the native methods 106. The macro instructions provided in native macro portion 108 operate to access, and if necessary, convert the Java parameters (e.g., Java primitive data types, Java reference objects). Accordingly, the native macro portion 108 operates to provide access and convert the parameters very efficiently. In addition, the native macro portion 108 can insulate the native methods 106 from the virtual machine internals.

As will appreciated by those skilled in the art, the lightweight native method invocation interface 104 can access the Java parameters directly. In comparison to conventional techniques, the lightweight native method invocation interface 104 can significantly reduce the overhead associated with invocation of native methods in Java programming environments.

In one preferred embodiment, the lightweight native method invocation interface 104 provides access to the Java parameters stored on the execution stack. To illustrate, Fig. 2 depicts a Java computing environment 200 in accordance to one embodiment of the invention. The Java computing environment 200 includes a Java execution stack 202 and a native execution stack 204. It should be noted that to invoke a native method, the stack frame associated with the method is placed on the execution stack 202. The stack frame can include parameters of the native method which represent one or more Java data types. Accordingly, parameter 1, parameter 2, ... parameter N, shown in Fig. 2, represent Java parameters associated with the native method. These parameters are converted into native parameter 1, native parameter 2, ... native parameter M which are placed on the native execution stack 204 to perform the processing of the native method written in another programming language.

As illustrated in Fig. 2, the lightweight Java native method invocation interface 206 accesses parameter reference 208. The parameter reference 208 is a reference to the parameter portion of the stack frame for the native method. Accordingly, parameter 1, parameter 2, ... parameter N of the native method can quickly be referenced by using the parameter reference 208 (i.e., use parameter reference [i] to access the i^{th} parameters). Using the parameter reference 208, these parameters can be accessed and converted by the lightweight Java method invocation interface 206 to an appropriate set of native parameters which are, in turn, placed on the native execution stack 204 (native parameter 1, native parameter 2, ...native parameter M). It should be noted that native parameter 1, native parameter 2, ... native parameter M are suitable for use by the native method written in a native programming language.

Fig. 3 illustrates a method for invocation of Java native methods in accordance with one embodiment of the invention. Initially, at operation 302, one or more Java parameters associated with a Java native method are placed on a Java execution stack. Next, at operation 304, a reference is provided to the one more or more Java parameters on the Java execution stack. The reference can be provided, for example, to one or more macros written in a native programming language (i.e., a programming language other than Java). Thereafter, at operation 306, at least one of the parameters on the Java execution stack is accessed using the provided reference. After the parameter has been accessed, at operation 308 the accessed parameter is converted to a native parameter. Thereafter, the native parameter is placed on the native execution stack at operation 310. Finally, at operation 312, the native method is executed using the native parameter on the native execution stack.

The many features and advantages of the present invention are apparent from the written description, and thus, it is intended by the appended claims to cover all such features and advantages of the invention. Further, since numerous modifications and changes will readily occur to those skilled in the art, it is not desired to limit the invention to the exact construction and

operation as illustrated and described. Hence, all suitable modifications and equivalents may be resorted to as falling within the scope of the invention.

What is claimed is: